Spring Boot实际应用讲解（四）：RESTful API


ZYRzyr (/u/f8ff63b17fc7) + 关注
2017.11.21 11:36* 字数 2036 阅读 361 评论 0 喜欢 5
(/u/f8ff63b17fc7)

> 文/ZYRzyr (https://www.jianshu.com/u/f8ff63b17fc7)
> 原文链接:http://www.jianshu.com/p/e907595e9d1d
> (https://www.jianshu.com/p/e907595e9d1d)

本文提纲

一、RESTful API通俗解释
二、为什么选择它
三、实例
四、最后

本文运行环境

```
Ubuntu 16.04 LTS
JDK 8 +
IntelliJ IDEA ULTIMATE 2017.2
Maven 3.5.0
Spring Boot 1.5.8.RELEASE
```

一、RESTful API通俗解释

RESTful API 即具有 REST 风格的 API，那什么是 REST 呢?
网上有很多介绍什么是 REST 的，但大多都是长篇大论的理论，即使有耐心读完，也不一定能真正理解其意。比如下面这一小句：

> REST全称Representational State Transfer，中文翻译【表述性状态传递】。

所以这个【表述性状态传递】到底是什么鬼？反正我是不明白。
经过自己长时间的使用与对标准化的追求，再加上平时的网络交互中，使用最多的是 HTTP 协议，所以自己总结出的在 HTTP 协议基础下的 REST 即：

> 客户端与服务器交互过程中，客户端用URL定位服务器资源，用一些动词（POST、GET、PUT、PATCH、DELETE等）描述对资源的操作，得到状态码判断操作结果如何。

实际项目中，很难写出**完全**满足 REST 标准的 API，所以此处只说一般情况下需要满足的点：

1.1 资源的定义

一般使用名词而不是动词来定义 URL，比如：
api.example.com/version/products 而不是 api.example.com/version/getProducts 表示获取商品。

1.2 使用合适的动词

GET ——获取资源
POST ——新建资源（很少情况下也可用于更新）
PUT ——更新资源（对整个资源的更新）
PATCH ——更新资源（对部分资源的更新，很少使用，某些类库不支持）
DELETE ——删除资源

1.3 使用标准的HTTP状态码

假设一种场景：有一个 API 是 POST 请求，请求参数是 name 和 age，其中 age 有一个后台验证，小于18则不添加并返回错误信息，当客户端调用此 API 时， age=11，此时不满足后台验证，返回了错误信息 必须是成年人，状态码是 200 .

到此，看似一切都很正常：请求成功，并且也有错误信息。但熟悉HTTP状态码的话就会发现：明明已经返回错误信息了，说明出错了，但还是返回了表示成功的 200，这让客户端在做处理时面临一种比较尴尬的情况：需要在 success 中写错误处理逻辑。

因此， RESTful API 中要求：当客户端请求时，达到预期目的才能返回 200，其它时候，根据具体的情况返回具体的状态码 (https://link.jianshu.com?t=http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml)，这样既标准，又能反应出问题出在何处。上面这种情况应该返回错误信息的同时，返回状态码 400，这样，客户端就能从自身找原因。

1.4 适当的表示

REST 并没有规定有何种方式来表示资源，但在 HTTP 协议下，一般的选择就是 JSON 或 XML，因此本文将以 JSON 表示这些资源。通俗点讲，就是请求一个 API 时得到的返回值中的数据的表现形式是 JSON 还是 XML 。

一般情况下，满足以上4点，即可称为 RESTful API，需要强调的是 REST 并不仅仅以上4点，想了解比较全面的 REST，可以看看这篇比较长的文章 (https://link.jianshu.com?t=https://www.cnblogs.com/loveis715/p/4669091.html)

二、为什么选择它

服务器提供的 API 可以有很多种形式，但为什么选择 RESTful API 呢？主要有以下两点：

2.1 统一的接口

在面对各种客户端存在的情况下： Web 、 Android， iOS 等， RESTful API 可以提供一套统一的接口为它们服务，另外，对于一些可以提供第三方服务的平台，如微信登录，支付宝支付等，它们并不需要显示的客户端存在，只需要一套提供服务的接口， RESTful API 就是很好的选择，并且 RESTful API 在微服务架构中也是常用的服务间通信方式。

2.2 对代码质量的追求

就像上一点说的统一接口，其它形式的 API 也能做到，比如：
只要服务器没有出系统问题，所有的请求都返回下面这种格式：

```
POST请求成功–无返回值
{
    "code": "1",
    "data": null,
    "message": "客户端请求成功"
}
```

```
GET请求成功–返回
{
    "code": "1",
    "data": [
        {
            "name": "Tom",
            "age": 12,
            "money": 100.5
        },
        {
            "name": "Bob",
            "age": 13,
            "money": 200.5
        }
    ],
    "message": "客户端请求成功"
}
```

```
请求失败–无返回值
{
    "code": "101",
    "data": null,
    "message": "密码错误"
}
```

这种形式，看起来似乎很标准，很合理，但其实它有几个坑：

1. 所有的 `code` 需要与客户端重新定义，哪个值表示什么意思，一旦有修改，需要重新规定，而 `RESTful API` 使用现成的 `HTTP` 状态码；
2. 如果 `data` 中的形式复杂，解析起来很麻烦，而 `RESTful API` 直接返回需要的数据，即使麻烦，始终少一层解析；
3. 客户端的所有响应逻辑全都写在 `success` 中，对于追求代码质量的人来说，不能容忍这种尴尬的局面，而 `RESTful API` 只要有错，就能在 `error` 中操作；
4. 看起来很专业，其实一点也不。

三、实例

`Spring Boot` 对于 `RESTful API` 也有很好的支持，本次实例继续使用前篇的项目，依次演示 `GET` 、 `POST` 、 `PUT` 、 `DELETE` ，4种请求方式。由于还没讲到数据库，所以暂时不涉及数据库的操作，只需关注上面第一大点中提到的4小点。

3.1 UserController

在上一篇文章中的 `UserController` 中新增以下方法，分别对应 `GET` 、 `POST` 、 `PUT` 、 `DELETE` ，4种请求方式：

```
    @GetMapping("/users")
    public ResponseEntity<List<User>> getUsers() {
        List<User> users = new ArrayList<>();
        User user1 = new User();
        user1.setName("Bob");
        user1.setAge(20);
        user1.setPassword("123456");

        User user2 = new User();
        user2.setName("Tom");
        user2.setAge(22);
        user2.setPassword("654321");

        //模拟从数据库取出数据
        users.add(user1);
        users.add(user2);
        return new ResponseEntity<>(users, HttpStatus.OK);
    }

    @PostMapping("/register")
    public ResponseEntity<String> register(@Valid User user, BindingResult bindingRe
        if (bindingResult.hasErrors()) {
            //如果验证出错，则返回错误信息，状态码400
            return new ResponseEntity<>(bindingResult.getFieldError().getDefaultMess
        }

        return new ResponseEntity<>("success", HttpStatus.OK);
    }

    @PutMapping("/profile")
    public ResponseEntity<User> updateUser(User newUser) {
        User oldUser = new User();
        oldUser.setName("old");
        oldUser.setAge(20);
        oldUser.setPassword("123456");

        //假设oldUser为数据库中已存在数据，用newUser更新oldUser
        if (!TextUtil.isEmpty(newUser.getName())) {
            oldUser.setName(newUser.getName());
        }

        if (newUser.getAge() != null) {
            oldUser.setAge(newUser.getAge());
        }

        if (!TextUtil.isEmpty(newUser.getPassword())) {
            oldUser.setPassword(newUser.getPassword());
        }

        return new ResponseEntity<>(oldUser, HttpStatus.OK);
    }

    @DeleteMapping("/user")
    public ResponseEntity deleteUser(String name) {
        //省略数据库删除数据操作
        return new ResponseEntity(HttpStatus.OK);
    }
```

1. 在 `UserController` 的类名上除了 `@RestController`， 还有一个 `@RequestMapping("/user")`， 表示 URL 是 `localhost:8080/user`；
2. `@GetMapping`， `@PostMapping`， `@PutMapping`， `@DeleteMapping`： 对应的该方法的访问方式分别是 `GET` 、 `POST` 、 `PUT` 、 `DELETE`， 其括号中的值接在上一点的后面，如：`localhost:8080/user/users`；
3. `ResponseEntity<T>`： 响应实体，其中可包含数据与状态码，或只包含状态码。

3.2 测试

3.2.1 单元测试

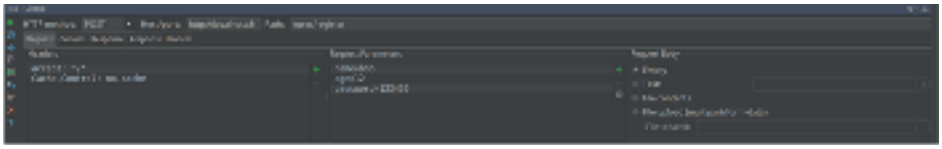
在 `UserControllerTest` 中添加如下测试用例：

```
    @Test
    public void testRegister_success() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.post("/user/register")
                .param("name", "Bob")
                .param("age", "20")
                .param("password", "123456"))
                .andExpect(MockMvcResultMatchers.status().isOk())
                .andExpect(MockMvcResultMatchers.content().string("success"));
    }

    @Test
    public void testRegister_age_error() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.post("/user/register")
                .param("name", "Bob")
                .param("age", "10")
                .param("password", "123456"))
                .andExpect(MockMvcResultMatchers.status().isBadRequest())//age=10，验
                .andExpect(MockMvcResultMatchers.content().string("必须是成年人"));
    }

    @Test
    public void testGetUsers() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get("/user/users"))
                .andExpect(MockMvcResultMatchers.status().isOk())
                .andExpect(MockMvcResultMatchers.content().json("[{\"name\":\"Bob\",
    }

    @Test
    public void testUpdateUser() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.put("/user/profile")
                .param("name", "new")
                .param("age", "30")
                .param("password", "555555"))
                .andExpect(MockMvcResultMatchers.status().isOk())
                .andExpect(MockMvcResultMatchers.content().json("{\"name\":\"new\",\
    }

    @Test
    public void testDelete() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.delete("/user/user"))
                .andExpect(MockMvcResultMatchers.status().isOk())
                .andExpect(MockMvcResultMatchers.content().string(""));
    }
```

运行，全部通过

3.2.2 IDEA REST Client

IDEA 自带 API 测试工具，打开 Tools -> Test RESTful Web Service：



REST Client.png

分别输入 HTTP method， Host/port， Path，若有请求参数，在 Request 的 Request Parameters 中添加参数，完成之后，点左边绿色第一个播放按钮，即可运行。
运行之后，可在 Response 中查看返回数据，在 Response Headers 中第一行查看状态码。

四、最后

本文先介绍了 RESTful API 的一些基本概念，后又在 Spring Boot 中实例演示。本次代码为了演示需要，在 UserController 中有很多重复代码，并且代码耦合严重，这些问题将会随着本系列文章的推出而逐步消除。

本文代码已上传至我的GitHub仓库 (https://link.jianshu.com?t=https://github.com/ZYRzyr/SpringBootDemo)，进入以后将branches (https://link.jianshu.com?t=https://github.com/ZYRzyr/SpringBootDemo/branches)切换为4-RESTful (https://link.jianshu.com?t=https://github.com/ZYRzyr/SpringBootDemo/tree/4-RESTful)即可看见。

前篇：
Spring Boot实际应用讲解（一）： Hello World (https://www.jianshu.com/p/60f7e025c680)
Spring Boot实际应用讲解（二）： 配置详解 (https://www.jianshu.com/p/d4c7f33c9b37)
Spring Boot实际应用讲解（三）： 表单验证 (https://www.jianshu.com/p/a2b4e61b5532)

后续将推出以下文章，敬请关注！

Spring Boot实际应用讲解（五）： AOP之请求日志 (https://www.jianshu.com/p/93216bf41182)
Spring Boot实际应用讲解（六）： MySQL + Spring-data-jpa(Hibernate) (https://www.jianshu.com/p/b204472d8126)
Spring Boot实际应用讲解（七）： 统一异常处理
Spring Boot实际应用讲解（八）： MySQL + Mybatis
Spring Boot实际应用讲解（九）： MySQL + Mybatis + Redis

文中若有错之处，还请各位批评指正，谢谢！

> 原文作者/ZYRzyr (https://www.jianshu.com/u/f8ff63b17fc7)
> 原文链接:http://www.jianshu.com/p/e907595e9d1d (https://www.jianshu.com/p/e907595e9d1d)

(https://link.jianshu.com?t=https://101709080007647.bqy.mobi)

(https://link.jianshu.com?t=https://101709080007647.bqy.mobi)

Spring Boot (/nb/18796030)

举报文章 © 著作权归作者所有

ZYRzyr (/u/f8ff63b17fc7) ♂
写了 17755 字，被 32 人关注，获得了 92 个喜欢
(/u/f8ff63b17fc7)

+关注

程序猿一枚 技能树加点情况： 移动端—Android——5/5 移动端—iOS——1/5 前端—JavaScript、HTML—…

喜欢的老铁，来一波关注666

赞赏支持

♡ 喜欢 (/sign_in?utm_source=desktop&utm_medium=not-signed-in-like-button) 5

更多分享